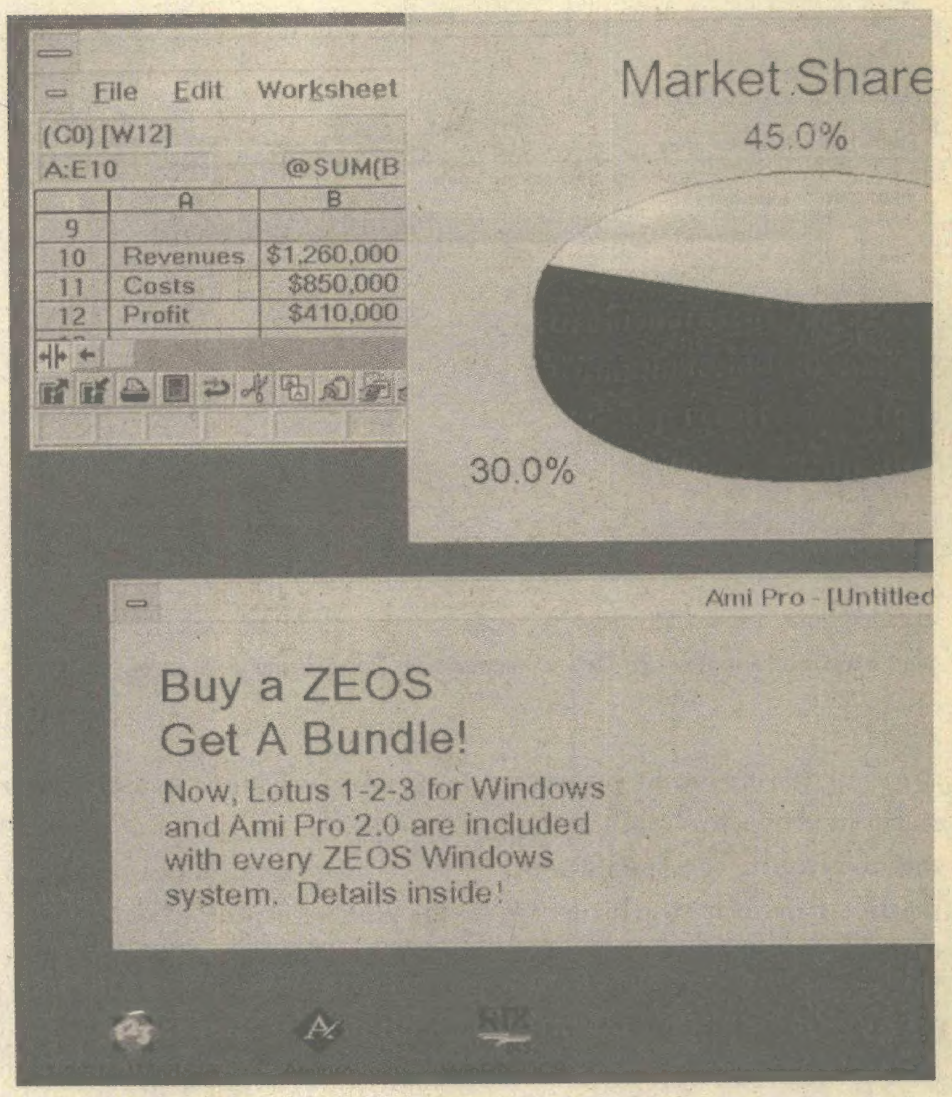
บุญเลิศ เอี่ยมทัศนา

เทอร์โบปาสคาลกับวินโดวส์ : หน้าต่าง

หลังจากได้แนะนำการติดตั้งและรันเทอร์โบปาสคาลบนวินโดวส์แล้ว ในบทนี้จะนำไปสู่การใช้งานหน้าต่าง



การเขียนโปรแกรมบนวินโดวส์ คือ การเขียนโปรแกรมที่จะใช้ (เพื่อให้ผู้ใช้ได้ใช้) สิ่งอำนวยความสะดวกที่จัดไว้ให้ในวินโดวส์ เช่น หน้าต่าง เมนู ไดอะล็อก เป็นต้น และสิ่งแรกที่จะต้องดำเนินการเพื่อเชื่อมโยงโปรแกรมของเรากับวินโดวส์คือ หน้าต่าง

โปรแกรมแรกบนวินโดวส์ : ใช้หน้าต่าง

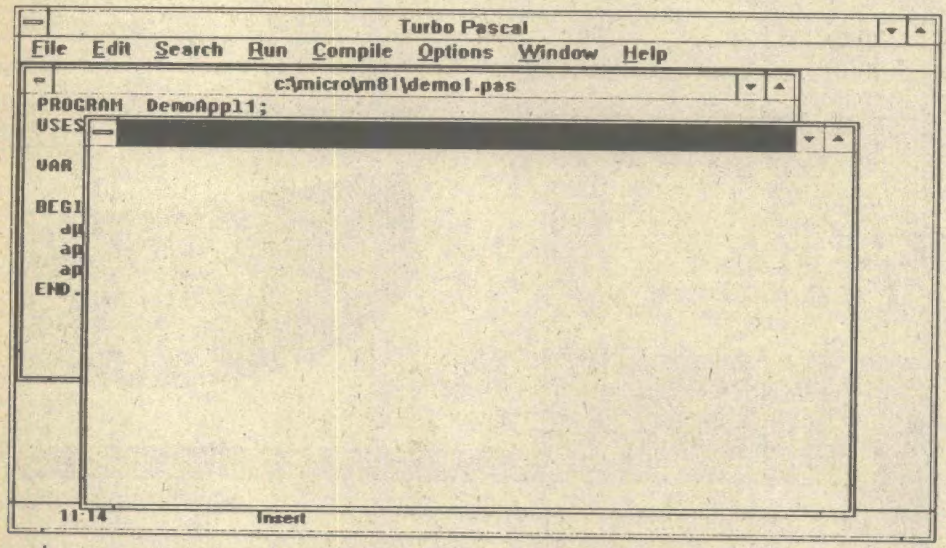
หลังจากการติดตั้งและรันเทอร์โบปาสคาลแล้วหลายท่าน (รวมทั้งผมด้วย) คงต้องการป้อนและทดลองรันโปรแกรมบนวินโดว์ เพื่อทำความรู้จักไว้ชั้นหนึ่งก่อน ขอให้ป้อนโปรแกรม DemoAppl1 ตามรูปที่ 1 และทดลองรันได้เลยครับ (เลือกรายการ Run/Run) จะปรากฏผลการรันดังรูปที่ 2

สำหรับท่านที่ยังไม่คุ้นเคยกับวินโดวส์ขอให้ทดลองใช้หน้าต่างนี้โดย ขยาย ย่อ ย้ายหน้าต่าง มินิไมซ์ (ย่อเล็กสุด) แมกซิไมซ์ (ขยายใหญ่สุด) และท้ายสุดคือ ปิดหน้าต่างนี้ด้วย Close (จะเป็นการยุติการรันของ DemoAppl1 ซึ่งถ้าใช้เม้าส์ขอให้กลับไปดูในฉบับที่ 79 หากใช้คีย์บอร์ดให้กดคีย์ Alt, space แล้วเลือกรายการด้วยคีย์ลูกศร ตามด้วย Enter ทั้งหมดนี้คือขีดความสามารถของหน้าต่างขั้นพื้นฐาน

แต่ถึงจะเป็นขั้นพื้นฐาน ผู้ที่เคยเขียนโปรแกรมเกี่ยวกับหน้าต่างมาก่อนทราบดีว่า จะต้องใช้เวลามากเพียงไร และยิ่งเพื่อให้ใช้งานได้กับจอภาพนานาชนิดด้วย วินโดวส์ทำให้ง่าย และเทอร์โบ-

รูปที่ 1 โปรแกรม DemoAppl1

```
PROGRAM  DemoAppl1;
USES  WObjects;

VAR    app : TApplication;

BEGIN
  app.Init('DemoAppl1');
  app.Run;
  app.Done;
END.
```



รูปที่ 2 ผลการรันโปรแกรม DemoAppl1 หน้าต่างของแอพพลิเคชันนี้ คือที่เห็นซ้อนทับ demo1.pas อยู่

ปาสคาลทำให้ง่ายยิ่งขึ้นอีก โดยที่โปรแกรมเดียวกันนี้หากเขียนด้วยภาษาซีจะยาวถึงประมาณ 80 บรรทัด ดังแสดงในรูปที่ 5 สำหรับคำอธิบายโปรแกรมนี้ขอพักไว้ก่อนครับ เพื่อแทรกเรื่องอื่น

แฮนเดิล (Handles)

ในระบบของวินโดวส์ เพื่อให้รันโปรแกรมในแบบมัลติทาสกิงได้ (เป็นเพียงเหตุผลหนึ่ง) ทั้งตัวโปรแกรมและข้อมูลจะถูกเปลี่ยนตำแหน่งในหน่วยความจำหลักอยู่ตลอดเวลา ในวินโดวส์จึงใช้พอยน์เตอร์ไปยังสิ่งต่าง ๆ เหล่านี้เป็นหลัก และทำตารางของพอยน์เตอร์ขึ้น มีอินเด็กซ์เรียกว่า แฮนเดิล เป็นค่า Integer ซึ่ง

- ถ้าเป็นออปเจ็กต์ของวินโดวส์เอง เช่นเมื่อเรากำหนดหน้าต่าง วินโดวส์จะกำหนดอินสแตนซ์ในฮีพและให้ค่าแฮนเดิลแก่ตัวแปร ซึ่งเราเพียงแต่นำตัวแปรนี้มาใช้ในการอ้างอิงเท่านั้น ดังจะเห็นในตัวอย่างต่อ ๆ ไป

- เมื่อเราจองหน่วยความจำในฮีพ (เช่นเมื่อใช้ New) วินโดวส์จะดำเนินการดังกล่าวมา แต่เราไม่ต้องสนใจและไม่ต้องรับทราบค่าแฮนเดิล คงใช้พอยน์เตอร์ตามปกติ นอกนั้นเป็นเรื่องของวินโดวส์กับเทอร์โบปาสคาล

สื่อความ (Message)

การเขียนโปรแกรมบนวินโดวส์มีเรื่องที่กลับกันกับที่เราเขียนโปรแกรมบนดอสคือ แทนที่เราจะเป็นฝ่ายเขียนโปรแกรมเรียกใช้โปรแกรมย่อยในวินโดวส์ฝ่ายเดียว วินโดวส์กลับเป็นฝ่ายเรียกใช้โปรแกรมย่อยของเราด้วย โดยส่งสื่อความมาให้ เช่น เมื่อผู้ใช้กดคีย์ วินโดวส์ก็จะส่งสื่อความกดคีย์มาให้ ซึ่งเราจะต้องเตรียมโปรแกรมย่อยสำหรับการนี้ไว้ให้

สื่อความที่วินโดวส์ส่งมาให้อยู่ในรูปของเรคอร์ด ประกอบด้วยฟิลด์ต่าง ๆ ดังรูปที่ 3 คือ

- hwnd เป็นแฮนเดิล (ค่าอินเด็กซ์ดังกล่าวมา) ของหน้าต่างของแอพพลิเคชันนั้น (ซึ่งอาจมีหลายหน้าต่าง)

- message เป็นหมายเลขสื่อความ (เป็นเลขจำนวนเต็ม Word) ซึ่งมีทั้งที่กำหนดไว้แล้วและให้ผู้เขียนโปรแกรมกำหนดใหม่ตามตารางในรูปที่ 4 ที่กำหนดไว้แล้วจะทำเป็นค่าคงตัวเพื่อไม่ต้องเป็นเลขจำนวน เช่น เมื่อผู้ใช้กดคีย์ซ้ายของเม้าส์จะได้สื่อความเป็น wm_LButtomDown เป็นต้น (wm ย่อมาจาก window message) ซึ่งเราไม่จำเป็นต้องทราบว่า wm_LButtomDown มีค่าเป็นเท่าไร

```
TYPE
    TMsg          = RECORD
      hwnd        : HWnd;
      message     : Word;
      wParam      : Word;
      lParam      : LongInt;
      time        : LongInt;
      pt          : TPoint;
                    END;
```

รูปที่ 3 แบบข้อมูลของ message ในวินโดวส์

ค่าคงตัว	ค่า	ช่วง	เป็นค่าของ
wm_First	$0000	$0000 - $7FFF	Window messages
wm_User	$0400	$0400 - $8FFF	User-definable message
id_First	$8000	$8000 - $8FFF	Child-ID messages
id_Internal	$8F00	$8F00 - $8FFF	Reserved for internal use
nf_First	$9000	$9000 - $9FFF	Notification message
nf_Internal	$9F00	$9F00 - $9FFF	Reserved for internal use
cm_First	$A000	$A000 - $FFFF	Command message
cm_Internal	$FF00	$FF00 - $FFFF	Reserved for internal use

รูปที่ 4 แสดงการจัดแบ่งค่าสื่อความ

- wParam (ย่อมาจาก Word Parameter) ขนาด 16 บิต ขึ้นอยู่กับแต่ละสื่อความ เช่นของ wm_LButtomDown จะแสดงว่ามีคีย์ Ctrl และ Shift กดพร้อมกันหรือไม่

- lParam (ย่อมาจาก Long Parameter) ขนาด 32 บิต ขึ้นอยู่กับแต่ละสื่อความเช่นกัน เช่นของ wm_LButtomDown จะแยกเป็นสองไบต์สูงกับสองไบต์ต่ำ โดยให้ค่าตำแหน่งของพอยน์เตอร์ของเม้าส์ในแกน x ที่สองไบต์ต่ำ และในแกน y ที่สองไบต์สูง

- time คือเวลาที่เก็บสื่อความลงคิว

- pt เป็นเรคอร์ดเก็บค่าของพอยน์เตอร์ของเม้าส์ขณะที่เก็บสื่อความลงคิว

สื่อความแบ่งออกเป็น 2 ชนิดคือ สื่อความที่อยู่ในคิวกับสื่อความที่ไม่อยู่ในคิว สื่อความที่อยู่ในคิวส่วนมากเป็นสื่อความที่เกิดจากการดำเนินการของผู้ใช้ เมื่อใช้คีย์และเม้าส์ นอกนั้นเป็นสื่อความที่ไม่อยู่ในคิว (มีข้อยกเว้นบ้าง) ซึ่งวินโดวส์ส่งให้แต่ละโปรแกรมย่อยโดยตรง

หมายเลขประจำตัว

นอกจากแฮนเดิลและสื่อความแล้ว ยังมีหมายเลขประจำตัวอีกกลุ่มหนึ่ง ซึ่งผู้เขียนโปรแกรมเป็นผู้กำหนดให้แก่ รีซอร์ส (resource : เช่น ไอคอน เมนู และฟอนต์ เป็นต้น) ดังจะกล่าวต่อไป นำมาเรียนให้ทราบตอนนี้เพื่อให้เห็นเปรียบเทียบกัน

เทียบกับโปรแกรมในภาษาซี

ผมคิดว่าการทำความเข้าใจกับการเขียนโปรแกรมบนวินโดวส์ด้วยเทอร์โบปาสคาล หากได้เห็นโปรแกรมในภาษาซีก่อนจะทำให้เข้าใจได้ง่าย เพราะจะได้เห็นส่วนที่เทอร์โบปาสคาลซ่อนไว้ในสกุล

คือโปรแกรม Hellowin ดังรูปที่ 5 ที่ผมใช้เปรียบเทียบกับโปรแกรมด้วยเทอร์โบปาสคาล ตามรูปที่ 1 ขอให้เทียบดูอีกครั้งครับโปรแกรมภาษาปาสคาลยาว 6 บรรทัด แต่โปรแกรมเดียวกันนี้ในภาษาซียาวประมาณ 80 บรรทัด (เมื่อรวมกับส่วนที่เกี่ยวกับการคอมพายล์อีก 2 ส่วน)

สำหรับท่านที่ไม่คุ้นเคยกับภาษาซีไม่เป็นไรครับ เพราะเราจะดูกันอย่างคร่าว ๆ เท่านั้น จะเห็นว่ามีโปรแกรมย่อยอยู่ 2 โปรแกรมคือ WinMain กับ WinProc ซึ่ง

WinMain

เทียบได้กับโปรแกรมหลักในภาษาปาสคาล (ในภาษาซีทั่วไปใช้คำว่า main แบ่งได้เป็น 3 ท่อนและมีหน้าที่ดังนี้คือ

1. ให้ค่ารายละเอียดแก่หน้าต่าง

ท่อนแรกเป็นท่อนที่ให้ค่ารายละเอียดแก่หน้าต่าง คือที่ให้แก่ wndclass ค่าที่สำคัญ คือ

- style เป็นค่าลักษณะของหน้าต่าง ค่าที่ให้ตามตัวอย่างคือ ให้ปรับขนาดของหน้าต่างใหม่เมื่อมีการเปลี่ยนแปลงเท่ากับให้ย่อ-ขยายขนาดของหน้าต่างได้

- lpfnWndProc เป็นพอยน์เตอร์ไปยังโปรแกรมย่อย ซึ่งค่าที่ให้คือ WndProc เป็นการเชื่อมต่อโปรแกรมนี้เข้ากับโปรแกรมหลัก (ผ่านทางวินโดวส์) ขอให้สังเกตครับ WndProc คือโปรแกรมย่อยในโปรแกรมนี้ แต่ในโปรแกรมหลักไม่ได้เรียกใช้โปรแกรมย่อยนี้เลย

- hInstance คือค่าแฮนเดิลของอินสแตนซ์ของโปรแกรมนี้คือ เมื่อเรา

รูปที่ 5 โปรแกรมในภาษาซี เพื่อดำเนินการเช่นเดียวกับ DemoApp1 (จาก Programming Windows โดย Charles Petzold)

```
/*  HELLOWIN  */

#include  <windows.h>

long FAR PASCAL WndProc(HWND, WORD, WORD, LONG);


int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
                   LPSTR lpszCmdParam, int nCmdShow)
{
static char szAppName[] = "HelloWin";
HWND        hwnd;
MSG         msg;
WNDCLASS    wndclass;

if  (!hPrevInstance)
  {
  wndclass.style         = CS_HREDRAW | CS_VREDRAW;
  wndclass.lpfnWndProc   = WndProc;
  wndclass.cbClsExtra    = 0;
  wndclass.cbWndExtra    = 0;
  wndclass.hInstance     = hInstance;
  wndclass.hIcon         = LoadIcon(NULL, ID_APPLICATION);
  wndclass.hCursor       = LoadCursor(NULL, IDC_ARROW);
  wndclass.hbrBackground = GetStockObject(WHITE_BRUSH);
  wndclass.lpszMenuName  = NULL;
  wndclass.lpszClassName = szAppName;

  RegisterClass(&wndclass);
  }

hwnd = CreateWindow(szAppName,          // window class name
                "The Hello Program",    // window caption
                WS_OVELAPPEDWINDOW,     // window style
                CW_USEDEFAULT,          // initial x position
                CW_USEDEFAULT,          // initial y position
                CW_USEDEFAULT,          // initial x size
                CW_USEDEFAULT,          // initial y size
                NULL,                   // parent window handle
                NULL,                   // window menu handle
                hInstance,              // program instance handle
                NULL);                  // creation parameter


ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

while(GetMessage(&msg,NULL,0,0))
  {
  TranslateMessage(&msg);
  DispatchMessage(&msg);
  }
return msg.wParm;
}
```

รูปที่ 5 (ต่อ)

```
long FAR PASCAL WndProc(HWND hwnd, WORD message,
                        WORD wParm, WORD lParm)
  {
  HDC         hdc;
  PAINTSTRUCT ps;

  switch(message)
    {
    case  WM_PAINT :
          hdc = BeginPaint(hwnd, &ps);
          EndPaint(hwnd, &ps);
          return 0;
    case  WM_DESTROY :
          PostQuitMessage(0);
          return 0;
    }
  return DefWindowProc(hwnd,message,wParm,lParm;
  }
```

รันโปรแกรมนี้แล้วยังไม่ยุติการรันแต่รันซ้ำใหม่ (บนวินโดวส์ทำได้) เป็นการรันโปรแกรมหลายก๊อบปี้ แต่ละก๊อบปี้คือ อินสแตนซ์หนึ่ง ๆ ซึ่งวินโดวส์จะให้ค่าแฮนเดิลต่าง ๆ กัน

- hbrBackground ให้ค่าไว้เป็น WHITE_BRUSH ทำให้เมื่อเพนต์ (ลบภาพในหน้าต่าง) จะให้พื้นหน้าต่างเป็นพื้นขาว

- lpszMenuName ให้ค่าชื่อของเมนูรีซอร์สคงต้องรอจนถึงเรื่องเมนูครับ ตอนนี้ให้เป็น NULL เพราะยังไม่มีเมนู

- lpszClassName เป็นชื่ออ้างอิง ส่วนมากจะให้ชื่อเดียวกับโปรแกรม

หลังจากให้ค่าต่าง ๆ แล้ว ได้ลงทะเบียนคือ Register Class ให้วินโดวส์นำไปเก็บไว้เป็นรายละเอียดของหน้าต่าง หากต้องการสร้างหลายหน้าต่าง (หน้าต่างต่อ ๆ ไปคือ หน้าต่างลูก) ให้ค่าใหม่ (ให้เฉพาะที่ไม่ตรงกัน) แล้วลงทะเบียนใหม่

2. สร้างหน้าต่าง

ท่อนต่อไปเป็นการสร้างหน้าต่างให้ปรากฏภาพบนจอภาพ ประกอบด้วยคำสั่ง CreateWindow เพื่อสร้างหน้าต่าง ShowWindow ให้แสดงหน้าต่าง ส่วน Update Window ให้แสดงภาพในหน้าต่างใหม่ แต่เนื่องจากยังไม่มีภาพใดในหน้าต่างจึงเพียงแต่ลบภาพในหน้าต่าง สำหรับ CreateWindow มีรายละเอียดควรทราบดังนี้คือ

- The Hello Program เป็นชื่อหน้าต่าง

- WS_OVERLAPPEDWINDOW เป็นค่ารูปแบบของหน้าต่าง ค่านี้ทำให้เกิดหน้าต่างดังรูปที่ 2 ซึ่งเป็นรูปแบบของหน้าต่างโดยทั่วไป

3. การทำงานกับวินโดวส์

การทำงานในส่วนนี้จะประสานการปฏิบัติกับวินโดวส์ กล่าวคือ

- GetMessage เพื่อขอสื่อความจากวินโดวส์เป็นการโอนการทำงานจากโปรแกรมไปให้วินโดวส์ และจะทำงานในวินโดวส์จนกว่าจะมีสื่อความส่งมาให้

- TranslateMessage เมื่อรับสื่อความมาแล้วจะส่งกลับไปให้วินโดวส์เพื่อให้ตีความคีย์บางคีย์ (แต่ส่งไปทุกสื่อความ)

- DispatchMessage เมื่อรับสื่อความกลับมาอีกครั้งแล้วจะส่งไปให้วินโดวส์อีก เพื่อให้ผ่านค่าไปให้แก่โปรแกรมย่อยของตน คือ WinProc แต่ด้วยวิธีเช่นนี้จึงทำให้กลายเป็นของวินโดวส์ไป ทั้งนี้เหตุผลหนึ่งคือ ให้โปรแกรมย่อยสามารถรับสื่อความโดยตรงจากวินโดวส์ได้ด้วย

จะออกจากจากวงรอบ while เมื่อสื่อความเป็น WM_QUIT คือเมื่อผู้ใช้ยุติการทำงานกับโปรแกรมนี้ ทำให้ GetMessage เป็นเท็จ

WndProc

ย่อมาจาก window procedure เป็นโปรแกรมย่อยที่เราเขียนขึ้นเพื่อให้

วินโดวส์เรียกใช้คือ เมื่อโปรแกรมหลักเรียกใช้ DispatchMessage วินโดวส์จะเรียกใช้โปรแกรมย่อยนี้ หรือเมื่อวินโดวส์เรียกใช้โดยตรง ด้วยการผ่านค่าในสื่อความมาให้ ดังที่เห็นที่ส่วนหัวของโปรแกรมย่อย ซึ่งจะนำมาตรวจและดำเนินการเช่นตามตัวอย่างนี้คือ เมื่อ message มีค่าเป็น

- WM_PAINT เป็นสื่อความเมื่อมีการเปลี่ยนแปลงขนาดหรือย้ายตำแหน่งของหน้าต่าง จุดประสงค์คือเพื่อให้แสดงภาพในหน้าต่างใหม่ แต่ในโปรแกรมนี้ไม่มีภาพในหน้าต่าง จึงเป็นการทำให้หน้าต่างว่าง

- WM_DESTROY เป็นสื่อความเมื่อผู้ใช้ต้องการปิดหน้าต่าง โดยให้ PostQuitMessage ดำเนินการ ซึ่งจะให้สื่อความ WM_QUIT แก่คิว ทำให้โปรแกรมหลักออกจากวงรอบ while ดังกล่าวมา

ท้ายของโปรแกรมย่อยได้เรียกใช้ DefWindowProc เพื่อให้ดำเนินการในสื่อความที่ตนไม่ได้ดำเนินการ เช่น สื่อความทั้งสองนี้อาจให้ DefWindowProc ดำเนินการได้เช่นกัน เพราะไม่มีสิ่งใดเป็นพิเศษ

Window procedure หรือที่นิยมเรียกกันสั้น ๆ ว่า วินโดว์พรอค ในความหมายว่าเป็นโปรแกรมย่อยของวินโดวส์จะใช้ชื่อใดก็ได้ และถ้ามีหน้าต่างลูกก็จะมีหลายโปรแกรมได้ หากการดำเนินการแตกต่างกัน สุดแต่จะให้แอดเดรสของโปรแกรมย่อยใดแก่ lpfnWndProc (lpfn ย่อมาจาก long pointer to function) ของหน้าต่างใด

เป็นมาตรฐาน

จะเห็นว่าโปรแกรมสำหรับแอพพลิเคชันบนวินโดวส์แบ่งออกเป็น 2 ส่วน เป็นมาตรฐานเดียวกัน คือ

- WinMain เป็นโปรแกรมหลักทำหน้าที่ในการกำหนดหน้าต่าง และประสานการติดต่อกับวินโดวส์ จะเขียนได้ในแบบเดียวกันนี้ในทุกแอพพลิเคชันแตกต่างกันน้อยมาก

- WndProc คือโปรแกรมย่อยของวินโดวส์ และส่วนนี้ครับคือส่วนที่เราจะเขียนโปรแกรมกัน ซึ่งเป็นการเขียนโปรแกรมให้วินโดวส์ และเพราะเหตุนี้กระมังครับจึงถือว่าเป็นการเขียนโปรแกรมในแบบโอโอพี เพราะตามโปรแกรมตัวอย่างภาษาซีที่ยกมามองไม่เห็นว่าเป็นโอโอพี

เป็นโอโอพีไม่เป็นโอโอพี

ที่ผมกล่าวว่ามองไม่เห็นว่าเป็นโอโอพี เป็นการมองจากภาษาที่เป็นโอโอแอล (OOL : Object-Oriented Language) เช่น เทอร์โบปาสคาล และ C++ ซึ่งจะต้องมีการกำหนดสกุล (class) เป็นรูปแบบแน่นอน

แต่ความจริงแล้วการเขียนโปรแกรมในแบบโอโอพีไม่จำเป็นต้องเขียนด้วยโอโอแอลเสมอไป สามารถเขียนได้แม้แต่ภาษาแอสเซมบลี โดยนำหลักการของโอโอพีมาใช้ เช่น

กำหนดข้อมูลเป็นเรคอร์ด กับจัดโปรแกรมย่อยสำหรับดำเนินการกับข้อมูลนี้โดยเฉพาะ และถือหลักปฏิบัติว่าจะเข้าถึงข้อมูลนี้ผ่านทางโปรแกรมย่อยที่จัดไว้เท่านั้น ก็จะเป็นการเขียนโปรแกรมในแบบโอโอพีตามหลักเอนแคปซูลแล้ว

สำหรับในเรื่องของวินโดวส์เป็นเรื่องที่ซ้อนกันอยู่พอสมควรคือ เมื่อให้ค่ารายละเอียดแก่หน้าต่างที่ให้แก่ wndclass ความจริงแล้วเป็นการสร้างรูปแบบ (template) เพราะเมื่อเรา CreateWindow วินโดวส์จะกำหนดเนื้อที่ในฮีพและนำค่านี้ไปให้ เรียกเป็นออปเจ็กต์

ส่วนโปรแกรมย่อยที่จะดำเนินการกับข้อมูลนี้วินโดวส์ได้จัดโปรแกรมย่อยส่วนหนึ่งไว้แล้ว เช่น การย่อ ขยาย และย้ายหน้าต่าง เป็นต้น อีกส่วนหนึ่งที่ให้เราเขียนใน WndProc จึงถือได้ว่าหน้าต่างเป็นออปเจ็กต์และเป็นการเขียนโปรแกรมในแบบโอโอพี

หากฉงนไม่ต้องทำความเข้าใจครับ ผมเขียนเผื่อไว้สำหรับท่านที่อ่านหนังสือในเรื่องการเขียนโปรแกรมบนวินโดวส์ด้วยภาษาซี ในหนังสือนั้นจะกล่าวถึงออปเจ็กต์และโอโอพี อาจจะไม่เข้าใจว่าเป็นออปเจ็กต์เป็นโอโอพีได้อย่างไร และทั้งนี้ขอให้ถือว่าเป็นการเปรียบเทียบไม่เกี่ยวกับเรื่องที่ผมจะกล่าวต่อไป

กับเทอร์โบปาสคาล

โดยที่เทอร์โบปาสคาลเป็นโอโอแอล จึงนำส่วนที่ควรจะเป็นออปเจ็กต์ของวินโดวส์มาทำเป็นออปเจ็กต์ โดยกำหนดเป็นสกุล มีสกุลที่สำคัญ 2 สกุล คือ

- TWindow โดยนำโปรแกรมและหน้าที่ของ WndProc (รูปที่ 2) มาให้แก่สกุลนี้ กับได้เพิ่มการสร้างหน้าต่าง

- TApplication ได้นำโปรแกรมและหน้าที่ของ WinMain เว้นการสร้างหน้าต่างมาให้แก่สกุลนี้ และมีอินสแตนซ์คือ พอยน์เตอร์ไปยังอิน-

```
TYPE
  DemoApp2C = OBJECT(TApplication)
    procedure  InitMainWindow; virtual;
            END;


(* ----------  DemoApp2C method  ------------- *)

PROCEDURE  DemoApp2C.InitMainWindow;
BEGIN
  MainWindow := New(PWindow, Init(nil, 'DEMO TWO'));
END;


(* ----------  MAIN PROGRAM  ---------------- *)

VAR   app :  DemoApp2C;
```

รูปที่ 6 โปรแกรม DemoAppl2

สแตนซ์ของสกุล TWindow อยู่ในสกุลนี้เช่นกัน คือเช่นเดียวกันกับที่ใน WinMain มีพอยน์เตอร์ไปยัง WndProc เป็นฟิลด์หนึ่ง

เรียกได้ว่าเป็นลอกแบบโปรแกรมในภาษาซีมาอย่างเกือบตรงตัว

คำอธิบายโปรแกรม

ถึงตรงนี้ผมคิดว่าคงอธิบายโปรแกรม DemoAppl1 ได้แล้วครับ คือ

- เริ่มด้วยกำหนดให้ app เป็นอินสแตนซ์ของสกุล TApplication ตามวิธีการเขียนโปรแกรมในแบบโอโอพี
- เมื่อ Init จะเป็นไปตามที่กล่าวใน WinMain ข้อ 1 และข้อ 2 ทุกประการ คือ TApplication จะให้ค่ารายละเอียดแก่หน้าต่าง แล้วให้ TWindow สร้างหน้าต่าง จะเห็นว่าเทอร์โบปาสคาลให้เราผ่านชื่ออ้างอิง (ClassName) ไปให้เท่านั้น เพราะค่ารายละเอียดอื่น (ตามที่ให้แก่ wndclass และ CreateWindow ในรูปที่ 5) เป็นค่าพื้นฐานจึงให้ค่าที่เตรียมไว้แล้ว
- Run ตรงกับการทำงานของวินโดวส์เป็นวงรอบใน WinMain ที่ว่าเป็นการประสานการปฏิบัติกับวินโดวส์
- Done เป็นเรื่องของการเขียนโปรแกรมในแบบโอโอพี ส่วนใหญ่คือเพื่อยกเลิกหน่วยความจำที่จองไว้

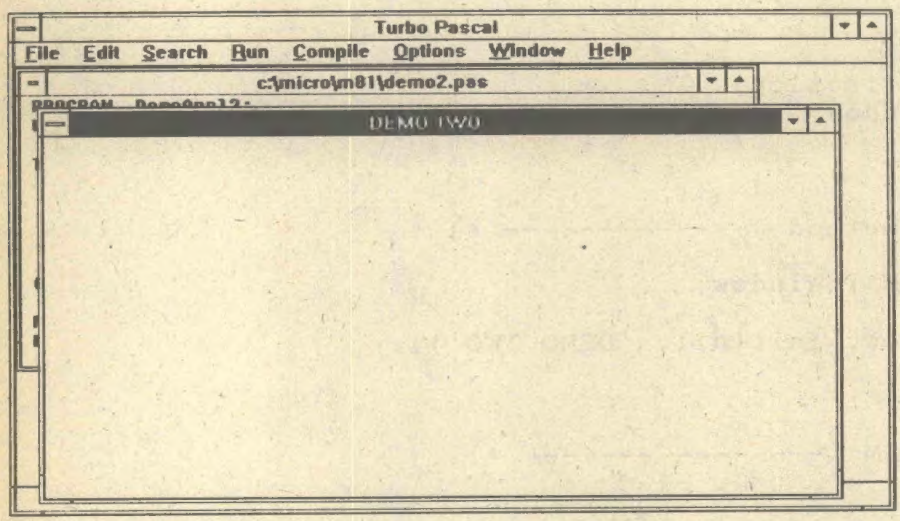
เช่นเดียวกับโปรแกรมในภาษาซี โปรแกรมหลักของเทอร์โบปาสคาลจะมีเพียงเท่านี้ ไม่ว่าจะเขียนแอพพลิเคชันในเรื่องใด ข้อแตกต่างจะอยู่นอกโปรแกรมหลักครับ

ให้ชื่อหน้าต่าง

เพราะผมต้องการแยกให้เห็นว่าหากเราเขียนโปรแกรมอย่างไร ไม่เขียนอย่างไร เทอร์โบปาสคาลจะดำเนินการอย่างไรให้แก่เราบ้าง แอพพลิเคชัน DemoAppl1 จึงไม่มีชื่อหน้าต่าง เพราะค่าพื้นฐาน (default) ที่เตรียมไว้ให้ชื่อหน้าต่างเป็นสตริงว่าง

แต่ทุกแอพพลิเคชันจำเป็นต้องให้ชื่อหน้าต่างของตน เพื่อให้ผู้ใช้ทราบว่ากำลังทำงานอยู่กับแอพพลิเคชันใด จะเขียนเป็นโปรแกรม DemoAppl2 ได้ดังรูปที่ 6

ผมไม่ได้แสดงโปรแกรมทั้งหมด



รูปที่ 7 ผลการรันโปรแกรม DemoAppl2

เพราะต้องการเน้นส่วนที่เพิ่มมา การป้อนโปรแกรมให้โหลด DEMO1.PAS แล้ว

- ลบบรรทัดที่กำหนดอินสแตนซ์ (คือ VAR)

- แทรกโปรแกรมตามรูปที่ 6 ไว้หน้า BEGIN ของโปรแกรมหลัก

- เก็บลงไฟล์โดยเลือกรายการ File/SaveAs และป้อนชื่อไฟล์ใหม่เป็น DEMO2.PAS แล้วจึงคอมพายล์และรัน

เมื่อรันจะได้ผลการรันดังรูปที่ 7 มีชื่อหน้าต่างเป็น DEMO ONE นั่นคือเป็นโปรแกรมที่เทียบเท่ากับโปรแกรมในภาษาซีที่ยกมาเป็นตัวอย่างอย่างแท้จริงครับ

สำหรับการเขียนโปรแกรมได้ดำเนินการดังนี้

- ได้สร้างสกุล DemoAppC เป็นผู้สืบสกุลของ TApplication ตามหลักการเขียนโปรแกรมในแบบโอโอพีเพื่อปรับปรุงสกุลเดิม แต่เทอร์โบปาสคาลนำมาใช้เพื่อให้ผู้เขียนโปรแกรมปรับแต่งโปรแกรม เช่น ในกรณีนี้คือเพิ่มชื่อหน้าต่าง

- กำหนดกิจกรรมทับ InitMainWindow ซึ่งมีหน้าที่เตรียม (Init) อินสแตนซ์ของสกุล TWindow โดยให้ค่า nil และ DEMO TWO ค่า nil หมายความว่าหน้าต่างนี้เป็นหน้าต่างแรกไม่มีหน้าต่างแม่ (parent) ส่วน DEMO TWO คือชื่อหน้าต่าง

ที่ว่าเขียนโปรแกรม

โปรแกรมทั้งสองดังกล่าวมาเป็นเพียงโครงของโปรแกรมที่เขียนขึ้นตามข้อกำหนดของเทอร์โบปาสคาลเท่านั้น ยังไม่ได้เขียนโปรแกรมของเราเลยครับ

โดยผมจะเขียนโปรแกรมให้ผู้ใช้ทำภาพดอกจันในหน้าต่างด้วยการกดปุ่มซ้ายของเม้าส์ และสมมติว่าเป็นงานที่มีการอ่านและเขียนกับไฟล์ ดังนั้นเพื่อให้เหมือนกับแอพพลิเคชันชั้นดีทั้งหลาย เมื่อผู้ใช้ยุติการรันจะตรวจสอบว่าได้มีการแก้ไขและเก็บลงไฟล์หรือยัง ถ้ายังจะถามว่าจะให้เก็บหรือไม่ ถ้าผู้ใช้ตอบ No ก็จะยุติการรัน ถ้าตอบ Yes ก็จะเก็บลงไฟล์ให้แล้วยุติการรัน

แต่นี่เป็นเพียงตัวอย่างง่าย ๆ เมื่อตอบ No จึงเพียงแต่ให้ยุติการรันเท่านั้น จะได้โปรแกรมตามรูปที่ 8 และผลการรันเมื่อผู้ใช้ต้องการยุติการรันตามรูปที่ 9 จุดประสงค์ของผมคือเพื่อแสดงกิจกรรม 2 ชนิดที่มีใน TWindow สำหรับโปรแกรมสามารถแบ่งออกได้เป็น 4 ส่วนคือ

ส่วนกำหนดสกุล

ส่วนแรกเป็นส่วนกำหนดสกุล ได้กำหนดสกุลไว้ดังนี้คือ

1. สกุล DemoApp3C เป็นไปตามที่กล่าวมา

2. ได้กำหนดสกุลใหม่คือ WinProC สืบสกุลจาก TWindow ซึ่งผมได้เคยกล่าวแล้วว่ามีอินสแตนซ์ของสกุลนี้อยู่ในสกุล TApplication และมีรายละเอียดดังนี้

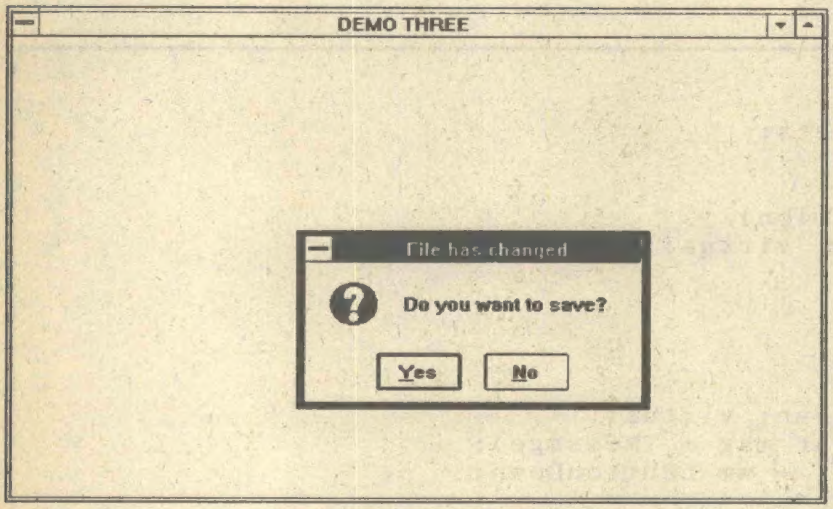
2.1 เมื่อเทียบกับซี WndProC ก็คือ WndProc ในรูปที่ 5 ที่เทอร์โบปาสคาลทำเช่นนี้ก็เพื่อใช้ประโยชน์ของการเขียนโปรแกรมในแบบโอโอพี โดยที่เทอร์โบปาสคาลได้กำหนดกิจกรรมที่จะดำเนินการกับสื่อความทุกสื่อความไว้ในสกุลนี้ ดังเช่นสื่อความ wm LButtomDown จะกำหนดเป็นกิจกรรม WMLButtomDown โดยตัดขีดล่างออก และให้เป็นเวอร์ชวล

2.2 เมื่อเราต้องการดำเนินการกับสื่อความใด ให้กำหนดกิจกรรมนั้นซ้ำ จะทำให้ย้ายการทำงานจากของเทอร์โบปาสคาลมาเป็นของเราตามหลักของเวอร์ชวล ดังเช่นตามตัวอย่างนี้คือ WMLButtomDown

2.3 สำหรับ CanClose เป็นกิจกรรมพิเศษ จะมีการเรียกใช้กิจกรรม

รูปที่ 8 โปรแกรม DemoAppl3

```
PROGRAM  DemoAppl3;
USES  WinTypes,WinProcs,WObjects;

TYPE
  DemoApplC = OBJECT(TApplication)
    procedure  InitMainWindow; virtual;
             END;

TYPE
  WndProCP = ^WndProC;
  WndProC = OBJECT(TWindow)
    function  CanClose : Boolean; virtual;
    procedure WMLButtonDown(var msg : TMessage);
              virtual wm_First + wm_LButtonDown;
               END;


(* ------------------  WndProC method  ---------------- *)

FUNCTION  WndProC.CanClose : Boolean;
VAR  reply : Integer;
BEGIN
  CanClose :=
    MessageBox(HWindow,'Do you want to save?',
      'File has changed', mb_YesNo or mb_IconQuestion)
           = id_No;
END;


PROCEDURE  WndProC.WMLButtonDown(var msg : TMessage);
VAR   dc : HDC;
BEGIN
  dc := GetDC(HWindow);
  TextOut(dc,msg.LParamLo,msg.LParamHi,'*',1);
  ReleaseDC(HWindow,dc);
END;


(* ------------------  DemoApplC method  ---------------- *)

PROCEDURE  DemoApplC.InitMainWindow;
BEGIN
  MainWindow := New(WndProCP, Init(nil,'DEMO THREE'));
END;

(* --------------------  MAIN PROGRAM  ------------------ *)

VAR   app :  DemoApplC;

BEGIN
  app.Init('DemoAppl3');
  app.Run;
  app.Done;
END.
```



รูปที่ 9 ผลการรันโปรแกรม DemoAppl3 เมื่อผู้ใช้ต้องการปิดหน้าต่าง DEMO THREE

นี้เมื่อผู้ใช้คลิ๊กซ้อนที่คอนโทรลเมนู หรือเลือกรายการ Control/Close เป็นการนำหลายสื่อความมารวมกัน โดยที่

- CanClose เป็นฟังก์ชันเพื่อสอบถามหน้าต่างว่าพร้อมที่จะปิดหรือไม่ หากให้ CanClose เป็นจริง แสดงว่าพร้อมที่จะปิด และวินโดวส์จะดำเนินการปิดหน้าต่างต่อไป

ส่วนโปรแกรม WndProC

ส่วนที่สองเป็นโปรแกรมของกิจกรรมใน WndProC กล่าวคือ

1. CanClose เพื่อการปิดหน้าต่าง ในโปรแกรมผมใช้ MessageBox ถามผู้ใช้ว่าจะให้เก็บข้อมูลลงไฟล์หรือไม่ นำคำตอบมาให้แก่ CanClose สำหรับ MessageBox เป็นเรื่องยาวพอสมควร จึงแยกไปเป็นหัวข้อต่างหาก

2. WMLButtomDown ทุกครั้งที่ผู้ใช้คลิ๊กเม้าส์ (กดปุ่มซ้ายของเม้าส์) จะมาทำงานที่โปรแกรมย่อยนี้ (ดังกล่าวมา) ผมจึงให้เอาค์พุตเครื่องหมายดอกจันณ ตำแหน่งของพอยน์เตอร์เมื่อคลิ๊กเม้าส์ ซึ่งการใช้ TextOut เราต้องเตรียม DC ก่อนและเมื่อเลิกใช้จะต้องปลดปล่อย (ReleaseDC) มีรายละเอียดของแต่ละเรื่องดังนี้

2.1 DC เป็นตัวแปรแบบ HDC (เทียบได้กับ Word ในเทอร์โบปาสคาล) ตั้งขึ้นเพื่อหมายถึง display context ซึ่งเป็นหน่วยความจำที่ใช้แทนพื้นของหน้าต่างเพื่อการวาดภาพ (การแสดงภาพอักขระถือเป็นการวาด) มีอยู่ด้วยกัน 5 หน่วย

2.2 คำสั่ง GetDC จะให้ค่าแฮนเดิลคือ หมายเลขของ display contact และเชื่อมต่อ display contact นี้เข้ากับหน้าต่าง

2.3 ค่าแฮนเดิลนี้ให้นำไปใช้กับโปรแกรมย่อยที่เกี่ยวกับการวาดภาพ เช่นที่ใช้กับ TextOut เมื่อใช้แล้วจะต้องปลดปล่อยไม่เช่นนั้นจะไม่พอใช้ทำให้เครื่องยุติการทำงาน

2.4 ค่าที่ผ่านไปให้แก่ TextOut มีตามลำดับดังนี้คือ ค่า DC ตำแหน่งของเม้าส์ในแกน x และ y สตริงข้อความที่จะให้นำออกแสดง และจำนวนอักขระในสตริง

ส่วนโปรแกรมของ DemoApplC

ส่วนที่สามเป็นโปรแกรมของกิจกรรมในสกุล DemoApplC เช่นเดียวกับโปรแกรมที่ผ่านมา แต่ให้ค่า WndProCP แทน PWindow เพื่อให้เป็นออปเจ็กต์ของสกุลใหม่นี้ เพื่อให้เรียกมายังกิจกรรมที่เขียนโปรแกรมให้ใหม่ใน WndProC แทนที่จะเป็น TWindow ของเทอร์โบปาสคาล

ส่วนโปรแกรมหลัก

ส่วนสุดท้ายเป็นส่วนของโปรแกรมหลัก ซึ่งจะเหมือนกันทุกแอพพลิเคชัน

โปรแกรมทั้งสี่ส่วนนี้ ส่วนที่เปลี่ยนแปลงในทุกแอพพลิเคชันคือส่วนที่สอง ซึ่งรวมทั้งการกำหนดสกุล WndProC ในส่วนที่หนึ่ง ผมจึงแยกเพื่อจะได้แทรกหรือถอนโปรแกรมได้โดยสะดวก เพื่อจะได้ไม่ต้องลงโปรแกรมทั้งหมดอีก

ชนิดของกิจกรรม

จากตัวอย่างนี้จะเห็นว่ามีกิจกรรมอยู่ 2 ชนิดที่นำมาใช้ในการเขียนโปรแกรมคือ

กิจกรรมสื่อความ

เป็นกิจกรรมที่เขียนขึ้นเพื่อสนองตอบสื่อความของวินโดวส์ รายการสื่อความดูได้จากคู่มือ การเขียนโปรแกรมจึงเป็นในลักษณะว่า ถ้ามีสื่อความนั้น ๆ เกิดขึ้นจะดำเนินการอย่างไร ซึ่งจะดำเนินการหรือไม่ก็ได้ แล้วแต่จุดประสงค์ของการเขียนโปรแกรม เช่น

จากตัวอย่างที่ยกมาได้เขียนโปรแกรมสนองตอบการกดปุ่มซ้ายของเม้าส์เช่นกัน ผู้ใช้อาจกดปุ่มขวาหรือกดคีย์บอร์ด แต่ไม่ได้นำมาดำเนินการก็จะไม่มีสิ่งใดเกิดขึ้น

กิจกรรมรวม

เป็นกิจกรรมเพื่อสนองตอบกรณีที่เกิดขึ้นจากหลายกรณี เช่น ตามตัวอย่างที่ยกมา จะมีการเรียกใช้ CanClose เมื่อผู้ใช้คลิ๊กซ้อนที่คอนโทรลเมนูหรือเลือกรายการ Control/Close กิจกรรมอย่าง CanClose เทอร์โบปาสคาลได้กำหนดไว้ในสกุล ในการเขียนโปรแกรมจึงต้องศึกษากิจกรรมในสกุล TWindow (ความจริงต้องศึกษาในทุกสกุล) แล้วเลือกมาดำเนินการคือนำมากำหนดกิจกรรมตามต้องการ ซึ่งถ้าไม่เลือกมาดำเนินการก็จะเป็นไปตามที่เขียนไว้ใน Twindow เช่น CanClose จะได้ค่าความเป็นจริงกลับไป คือให้ปิดหน้าต่างได้

กรอบสื่อความ (Message box)

กรอบสื่อความเป็นกรอบที่จะแสดงข้อความให้รับทราบ ซึ่งเราสามารถให้ผู้ใช้รับทราบเพียงอย่างเดียวหรือตอบปฏิเสธได้ด้วย หรืออย่างอื่นอีก เป็นหน้าต่างชนิดหนึ่งและเป็นหน้าต่างที่สร้างได้ง่ายที่สุดเพียงแต่เรียกใช้ฟังก์ชัน MessageBox เท่านั้น ตามส่วนหัวดังรูปที่ 10

```
MessageBox (Parent : HWnd ;
            Txt, Caption : PChar ;
            TextType : Word) : Integer;
```

รูปที่ 10

โดยที่

- Parent คือ ค่าแฮนเดิลของหน้าต่างหลักของ app (ท้ายของรูปที่ 8) ซึ่งวินโดวส์ได้ให้ค่าไว้ในตัวแปร HWindow อยู่แล้ว เราเพียงแต่นำมาอ้างอิง

- Txt คือข้อความที่จะให้แสดงในหน้าต่าง

- Caption คือข้อความที่จะให้แสดงที่กรอบของหน้าต่าง ซึ่งโดยปกติคือชื่อหน้าต่าง

- TextType หมายถึงแบบของข้อความแต่ความหมายที่แท้ คือจะให้แสดงปุ่มอะไรบ้าง (เช่น OK กับ Cancel) และให้แสดงไอคอนอะไรด้วยหรือไม่ อย่างเช่นตามรูปที่ 9 ให้แสดงภาพไอคอน? ค่าที่ให้นี้เป็นเลขจำนวนซึ่งได้ทำเป็นค่าคงตัวไว้ดังรูปที่ 11 และเป็นค่าบิตจึงนำมา OR กันได้

```
   mb_AbortRetryIgnore     Abort,Retry and Ignore buttons
   mb_ApplModal            Modal message box ( default )
   mb_DefButton1           First botton = default ( default )
   mb_DefButton2           Second botton = default
   mb_DefButton3           Third botton = default
   mb_IconAsterisk         The 'i' icon
   mb_IconExclamation      The '!' icon
   mb_IconHand             The stop sign icon
   mb_IconInformation      The 'i' icon
   mb_IconQuestion         The '?' icon
   mb_IconStop             The stop sign icon
   mb_OK                   OK button
   mb_OKCancel             OK and Cancel buttons
   mb_RetryCancel          Retry and Cancel buttons
   mb_SystemModal          For potentially damaging situations
   mb_TaskModal            For no parent window
   mb_YesNo                Yes and No buttons
   mb_YesNoCancel          Yes,No and Cancel
=================================================================
   mb_DefMask      mb_DefButton1,mb_DefButton2 and mb_DefButton3
   mb_IconMask     mb_IconASterisk,mb_IconExclamation,
                   mb_IconHand,mb_IconInformation,
                   mb_IconQuestion,mb_IconStop
   mb_ModeMask     mb_ApplModal,mb_SystemModal,mb_TaskModal
   mb_TypeMask     mb_AbortRetryIgnore,mb_OK,mb_OKCancel
                   mb_RetryCancel,mb_YesNo,mb_YesNoCancel
```

รูปที่ 11 แสดงรายการค่าคงตัวที่จะให้แก่ Text_Type ด้านซ้ายมือค่าคงตัวด้านขวาคือความหมายที่จะให้แสดงปุ่มและไอคอนต่าง ๆ ใน message box

```
id_Abort          id_Cancel
id_Ignore         id_No
id_OK             id_Retry
id_Yes
```

รูปที่ 12 ค่าที่ฟังก์ชัน MessageBox จะส่งกลับมา ตามที่ผู้ใช้จะเลือกปุ่มใด

สำหรับ MessageBox จะคืนค่ามาเป็นเลขจำนวนเต็มเช่นกัน เป็นค่าอันเกิดจากการดำเนินการของผู้ใช้ ได้ทำเป็นค่าคงตัวไว้ตามรูปที่ 12 ใช้เป็นเงื่อนไขของการเขียนโปรแกรมต่อไป

โปรแกรม 3 รูปแบบ

การเขียนโปรแกรมบนวินโดวส์ด้วยเทอร์โบปาสคาลตามที่กล่าวมานี้ สามารถเขียนได้ 3 วิธี คือ

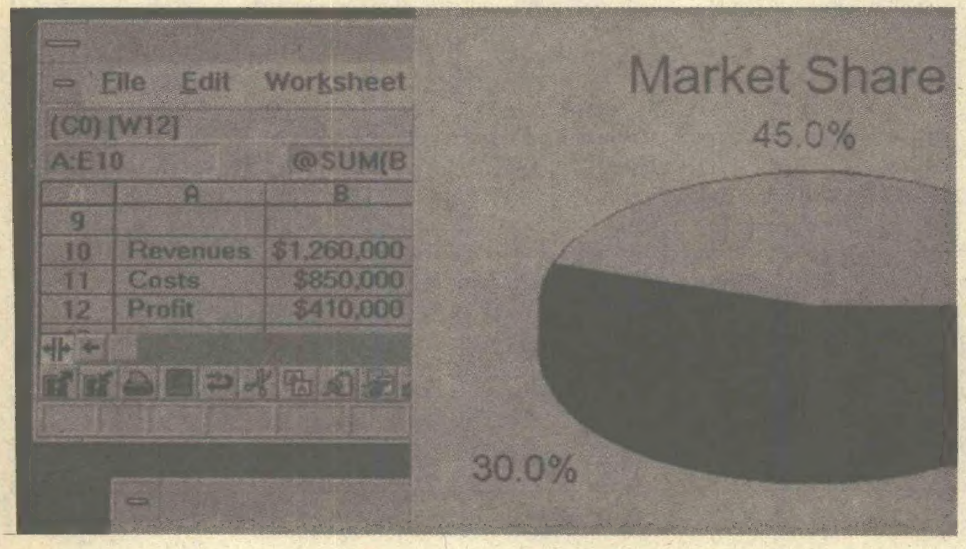
1. ในแบบโอโอพี คือการเขียนโปรแกรมโดยใช้สกุลที่มีอยู่ในเทอร์โบปาสคาล ดังที่นำมาใช้แล้ว 2 สกุล ในโปรแกรม DemoAppl3 เทอร์โบปาสคาลรวมเรียกสกุลในกลุ่มนี้ ซึ่งเกี่ยวกับหน้าต่างว่า ObjectWindow

2. เขียนเลียนแบบภาษาซี เป็นการเขียนเลียนแบบโปรแกรมบนวินโดวส์ที่เขียนด้วยภาษาซี ดังเช่นโปรแกรมในภาษาซีตามรูปที่ 5 ผมได้นำมาเขียนใหม่ด้วยเทอร์โบปาสคาลตามรูปที่ 13 ที่ผมว่าเลียนแบบเพราะการเขียนโปรแกรมบนวินโดวส์เริ่มด้วยภาษาซีก่อน

3. โดยใช้ยูนิต WinCrt เป็นแบบของเทอร์โบปาสคาลโดยเฉพาะ ดังที่กล่าวในตอนเตรียมการ (2) ฉบับที่ 80 ซึ่งทำให้สามารถอินพุตและเอาต์พุตด้วย Read และ Write (เช่นการเขียนโปรแกรมบนดอส) ได้ และให้สามารถใช้หน้าต่างกับสครอลบาร์ได้ โดยหลักใหญ่คือเพื่อให้ดูผลการรันได้ทั้งหมด แต่ใช้สิ่งอำนวยความสะดวกอื่น ๆ ของวินโดวส์ไม่ได้

การเขียนโปรแกรมทั้ง 3 วิธีนี้ที่เทอร์โบต้องการให้ใช้คือ การเขียนในแบบโอโอพี สำหรับการเขียนเลียนแบบภาษาซียังไม่มีเหตุผลใดที่จะใช้วิธีนี้ ตรงกันข้ามผู้ที่ใช้ภาษาซีอาจเขียนด้วยเทอร์โบปาสคาลในแบบโอโอพีก่อน เพราะจะพัฒนาโปรแกรมได้เร็วกว่ากัน เมื่อโปรแกรมทำงานได้ดีแล้วจึงแปลงเป็นภาษาซี

ส่วนการเขียนโดยใช้ยูนิต WinCrt อาจจะเหมาะกับงานบางงานที่มีอินพุตเอาต์พุตโดยเฉพาะเอาต์พุตกับจอภาพเป็นจำนวนมาก ซึ่งจะมีน้อย



ก่อนเดินทางต่อ

ก่อนที่จะจบตอนนี้เพื่อไปต่อในตอนหน้า ผมมีเรื่องที่จะขอนำมากล่าว คือ การเขียนโปรแกรมบนวินโดวส์เป็นทั้งเรื่องที่ใหม่ (ต่อผม) และมีเรื่องที่จะนำมากล่าวถึงมากมาย สิ่งที่ผมจะนำมาเสนอคือการใช้สิ่งอำนวยความสะดวกของวินโดวส์ในโปรแกรมของเรา ดังเช่นตอนนี้ได้กล่าวถึงหน้าต่างและกรอบสื่อความ

เป็นการกล่าวถึงสิ่งอำนวยความสะดวกในแต่ละเรื่องเป็นเรื่อง ๆ ไป ไม่ค่อยจะสัมพันธ์กัน เปรียบเสมือนกล่าวถึงการสร้างรูปต่อให้คุณไปสร้างแล้วนำไปต่อเป็นโปรแกรมด้วยตนเอง ซึ่งที่จริงแล้วถ้าคุณรู้จักสิ่งอำนวยความสะดวกของวินโดวส์ทั้งหมดก็เท่ากับรู้จักวิธีการเขียนโปรแกรมบนวินโดวส์ครับ แต่ผมจะพยายามหาโปรแกรมที่เป็นเรื่องราวมาแสดงด้วย

สำหรับเนื้อหาของบทความ ผมนำมาจากคู่มือเทอร์โบปาสคาล โดยนำมาปรับแต่งตามที่เห็นสมควร ดังนั้นหากต้องการสามารถเดินหน้าไปก่อนได้ พบกันตอนต่อไปในเรื่องของเมนูครับ

รูปที่ 13 โปรแกรมในภาษาปาสคาลที่เขียนเลียนแบบโปรแกรมในภาษาซีรูปที่ 6

```
PROGRAM  HelloWindows;
USES  WinTypes, WinProcs;


FUNCTION WndProc (hanwnd : HWnd; message: Word;
                  wParam: Word; lParam: LongInt): LongInt; export;
VAR  handc : HDC;  ps : TPaintStruct;
BEGIN
  WndProc := 0;
  case Message of
    wm_Paint : begin  handc := BeginPaint(hanwnd,ps);
                      EndPaint(hanwnd,ps)  end;
    wm_Destroy : PostQuitMessage(0);
  else
    WndProc := DefWindowProc(hanwnd,message,wParam,lParam);
  end;
END;


CONST
  szAppName = 'HelloWin';
VAR
  hanwnd     : HWnd;
  msg        : TMsg;
  wndcls     : TWndClass;

BEGIN   { MAIN }
  if hPrevInst = 0 then
  begin
    wndcls.style := cs_HRedraw or cs_VRedraw ;
    wndcls.lpfnWndProc := @WndProc;
    wndcls.cbClsExtra := 0;
    wndcls.cbWndExtra := 0;
    wndcls.hInstance := hInstance;
    wndcls.hIcon := LoadIcon(0, 'ide_SimplePaint');
    wndcls.hCursor := LoadCursor(0, idc_Arrow );
    wndcls.hbrBackground := GetStockObject(White_Brush);
    wndcls.lpszMenuName := szAppName;
    wndcls.lpszClassName := szAppName;

    if not RegisterClass(wndcls) then Halt;
  end;

  { Create and display the window. }
  hanwnd := CreateWindow(szAppName,'The Hello Program',
    ws_OverLappedWindow, cw_UseDefault, 0,
    cw_UseDefault, 0, 0, 0, hInstance, nil);

  ShowWindow(hanwnd, CmdShow);
  UpdateWindow(hanwnd);

  while GetMessage(msg, 0, 0, 0)  do
  begin
    TranslateMessage(msg);
    DispatchMessage(msg);
  end;
END.
```